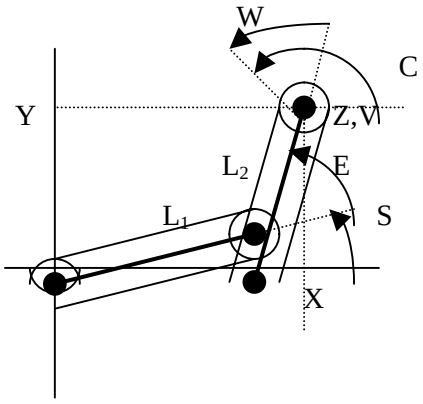
SCARA Robot Kinematics

A 4-axis SCARA (Selective Compliance Assembly Robot Arm) robot has parallel shoulder, elbow, and wrist rotary joints, and a linear vertical axis through the center of rotation of the wrist. This type of manipulator is very common in light-duty applications such as electronic assembly. Most users want to program this device as an XYZ Cartesian system with a rotary angle (C) for the tool, leaving the controller to compute the required joint angles and lengths. Turbo PMAC's integrated kinematics algorithms make this easy.

Mechanism Description

In this example, the upper-arm length (L_1) is 400 mm, and the lower-arm length (L_2) is 300 mm. The shoulder joint (S), the elbow joint (E), and the wrist joint (W) have resolutions of 1000 counts per degree. Rotation in the positive direction for all 3 joints is counter-clockwise when viewed from the top. The vertical axis (V) has a resolution of 100 counts per millimeter, and movement in the positive direction goes up. When the shoulder, elbow, and wrist joints are at their zero-degree positions, the two links are both extended along the X-axis and the tool orientation C is at zero degrees. When the vertical axis is at its home position, it is 250 mm above the Z-axis zero point. Due to wiring constraints, rollover of the rotary axes is not permitted.

This diagram shows a top view of the mechanism. Z-axis motion is into and out of the page.



Forward Kinematics

The forward-kinematic equations are:

$$X = L_1 \cos(S) + L_2 \cos(S + E)$$

$$Y = L_1 \sin(S) + L_2 \sin(S + E)$$

$$C = S + E + W$$

$$Z = V + Z_0$$

To implement these equations in a Turbo PMAC forward-kinematic program for Coordinate System 1 that converts the shoulder angle in Motor 1, the elbow angle in Motor 2, the wrist angle in Motor 3, and the vertical position in Motor 4, to the X, Y, and Z tip coordinates in millimeters, and the tip angle C in degrees, the following setup and program could be used:

Macro Substitution Definitions

Status/control bits using suggested M-variable definitions

```
#define Mtr1Homed  M145          ; Motor 1 home complete bit
Mtr1Homed->Y:$0000C0,10,1        ; Bit 10 in Motor 1 status word
#define Mtr2Homed  M245          ; Motor 2 home complete bit
Mtr2Homed->Y:$000140,10,1        ; Bit 10 in Motor 2 status word
#define Mtr3Homed  M345          ; Motor 3 home complete bit
Mtr3Homed->Y:$0001C0,10,1        ; Bit 10 in Motor 3 status word
```

```
#define Mtr4Homed  M445          ; Motor 4 home complete bit
Mtr4Homed->Y:$000240,10,1        ; Bit 10 in Motor 4 status word
#define CS1RunTimeErr  M5182     ; CS 1 run-time error bit
CS1RunTimeErr->Y:$00203F,22,1    ; Bit 22 in C.S. 1 status word
```

Definitions to Variables with Fixed Functions

```
#define Mtr1KinPos  P1           ; #1 pos in cts for kinematics
#define Mtr2KinPos  P2           ; #2 pos in cts for kinematics
#define Mtr3KinPos  P3           ; #3 pos in cts for kinematics
#define Mtr4KinPos  P4           ; #4 pos in cts for kinematics
#define XkinPos  Q7              ; X-axis pos in mm for kin
#define YkinPos  Q8              ; Y-axis pos in mm for kin
#define ZkinPos  Q9              ; Z-axis pos in mm for kin
#define CkinPos  Q3              ; C-axis pos in deg for kin
```

Definitions to Variables with Open Functions

```
#define Len1  Q81                ; Shoulder-to-elbow length
#define Len2  Q82                ; Elbow-to-wrist length
#define Ks  Q83                  ; Shoulder counts/degree
#define Ke  Q84                  ; Elbow counts/degree
#define Kw  Q85                  ; Wrist counts/degree
#define Kv  Q86                  ; Vertical counts/degree
#define Z0  Q87                  ; Z-axis offset
#define Spos  Q88                ; Shoulder position in deg
#define Epos  Q89                ; Elbow position in deg
#define Wpos  Q90                ; Wrist position in deg
```

Setup for Program

```
I15=0                            ; Trig calcs in degrees
I5150=1                          ; Enable kinematics for C.S.1
```

Set System Constants

```
Len1=400                         ; L1 (mm)
Len2=300                         ; L2 (mm)
Ks=1000                          ; Counts per degree for shoulder
Ke=1000                          ; Counts per degree for elbow
Kw=1000                          ; Counts per degree for wrist
Kv=100                           ; Counts per mm for vertical
Z0=250                           ; Z-axis offset in mm
```

Forward-Kinematic Program Buffer

```
&1 OPEN FORWARD                  ; Forward kinematics for CS 1
CLEAR                            ; Erase existing contents
; Check if motor positions properly referenced
IF (Mtr1Homed=1 AND Mtr2Homed=1 AND Mtr3Homed=1 AND Mtr4Homed=1)
  SPos=Mtr1KinPos/Ks             ; Shoulder angle in degrees
  EPos=Mtr2KinPos/Ke             ; Elbow angle in degrees
  WPos=Mtr3KinPos/Kw             ; Wrist angle in degrees
  XKinPos=Len1*COS(SPos)+Len2*COS(SPos+EPos)    ; Tool-tip X
  YKinPos=Len1*SIN(SPos)+Len2*SIN(SPos+EPos)    ; Tool-tip Y
  CKinPos=SPos+EPos+WPos         ; Tool-tip angle
  ZKinPos=Mtr4KinPos/Kv+Z0       ; Tool-tip height
ELSE                             ; Not referenced; halt operation
  CS1RunTimeErr=1                ; Set run-time error bit
ENDIF
CLOSE
```

This forward-kinematics program will be executed automatically every time Coordinate System 1 starts a motion program. It can also be executed on a **PMATCH** command.

Inverse Kinematics

Limiting ourselves to positive values of the elbow (E) angle, producing the right-armed case (done by selecting the positive arc-cosine solutions), we can write our inverse kinematic equations as follows:

To implement these equations in a Turbo PMAC inverse-kinematic program for Coordinate System 1 that converts the X, Y, and Z tip coordinates in millimeters and the tip angle C in degrees to the shoulder

$$E = +\cos^{-1}\left(\frac{X^2+Y^2-L_1^2-L_2^2}{2L_1L_2}\right)$$

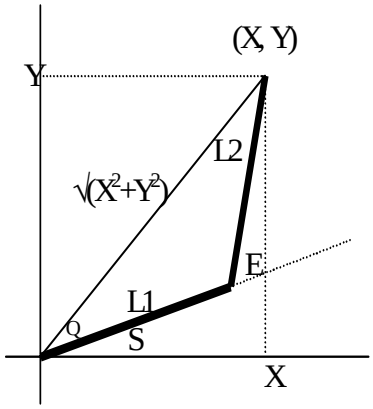
$$S+Q = \arctan 2(Y,X)$$

$$Q = +\cos^{-1}\left(\frac{X^2+Y^2+L_1^2-L_2^2}{2L_1\sqrt{X^2+Y^2}}\right)$$

$$S = (S+Q)-Q$$

$$W = C-S-E$$

$$V = Z-Z_0$$



angle in Motor 1, the elbow angle in Motor 2, the wrist angle in Motor 3, and the vertical position in Motor 4, the following program could be used. System constants are the same as for the above forward kinematic program.

Additional Macro Substitution Definitions

```
#define CosTerm  Q0              ; 2nd arg for ATAN2
#define SumLenSqrd  Q93          ; L1^2 + L2^2
#define ProdOfLens  Q94          ; 2*L1*L2
#define DifLenSqrd  Q95          ; L1^2 – L2^2
#define Temp1  Q96
#define Temp2  Q97
#define SplusQ  Q98              ; Shoulder-to-wrist angle (deg)
#define Qpos  Q99                ; Angle inside triangle (deg)
```

Additional Setup for Inverse Kinematic Program

```
&1
#1->I                        ; Motor 1 assigned to inverse kinematic axis in CS 1
#2->I                        ; Motor 2 assigned to inverse kinematic axis in CS 1
#3->I                        ; Motor 3 assigned to inverse kinematic axis in CS 1
#4->I                        ; Motor 4 assigned to inverse kinematic axis in CS 1
I5113=10                     ; Solve inv. kin. every 10 msec along path
```

Pre-compute Additional System Constants

```
SumLenSqrd=Len1*Len1+Len2*Len2
ProdOfLens=2*Len1*Len2
DifLenSqrd=Len1*Len1-Len2*Len2
```

Inverse-Kinematic Algorithm to be Executed Repeatedly

```
&1 OPEN INVERSE                        ; Inverse kinematics for CS 1
CLEAR                                  ; Erase existing contents
Temp1=XKinPos*XKinPos+YKinPos*YKinPos ; X^2+Y^2
Temp2=(Temp1-SumLenSqrd)/ProdOfLens    ; cos(E)
IF (ABS(Temp2)<0.996)                  ; Valid soln w/ 5 deg margin?
  EPos=ACOS(Temp2)                     ; Elbow angle (deg)
  CosTerm=XKinPos                      ; X into cos argument for ATAN2
  SPlusQ=ATAN2(YKinPos)                ; S+Q = ATAN2(Y,X)
```

```
  QPos=ACOS((Temp1+DifLenSqrd)/(2*Len1*SQRT(Temp1))) ; Q (deg)
  SPos=SPlusQ-QPos                    ; Shoulder angle (deg)
  Mtr1KinPos=SPos*Ks                  ; Convert to counts
  Mtr2KinPos=EPos*Ke                  ; Convert to counts
  Mtr3KinPos=(CKinPos-SPos-EPos)*Kw   ; Wrist position in counts
  Mtr4KinPos=(ZKinPos-Z0)*Kv          ; Vertical position in counts
ELSE                                  ; Not valid, halt operation
  CS1RunTimeErr=1                     ; Set run-time error bit
ENDIF
CLOSE
```

The inverse-kinematic program is run once per move segment (every I5113 msec) for LINEAR and CIRCLE-mode moves; it is run once per programmed move for other move modes. Note that this example checks for illegal (out-of-range) move commands by checking to see if the cosine of the elbow angle is in legal range. Other checks are possible as well.

Extension: Tip-Mode and Joint-Mode Programming

This following extension to the above example permits both tip-mode and joint-mode programming. In joint-mode programming, the kinematics algorithms are really pass-throughs, just scaling positions between raw counts and degrees.

In joint mode, it is possible to switch between the right-armed case and the left-armedcase. Once back in tip mode, the arm will stay in that orientation. In the inverse kinematic algorithm, the forward kinematics for the mode not used are computed each cycle so the mode can be switched properly at any time.

This extension uses the setup and definitions from the above example, and adds a few more.

Additional Substitutions and Definitions

```
#define AkinPos  Q1            ; A-axis pos in deg for kin
#define BkinPos  Q2            ; B-axis pos in deg for kin
#define WkinPos  Q6            ; W-axis pos in deg for kin
#define SgnAcos  Q91           ; Determines left from right
```

Forward-Kinematic Program Buffer

```
&1 OPEN FORWARD                ; Forward kinematics for CS 1
CLEAR                          ; Erase existing contents
; Check if motor positions properly referenced
IF (Mtr1Homed=1 AND Mtr2Homed=1 AND Mtr3Homed=1 AND Mtr4Homed=1)
  ; Compute scaled joint positions for joint-mode programming
  AKinPos=Mtr1KinPos/Ks
  BKinPos=Mtr2KinPos/Ke
  WKinPos=Mtr3KinPos/Kw
  ; Compute tool-tip positions for tip-mode programming
  XKinPos=Len1*COS(AKinPos)+Len2*COS(AKinPos+BKinPos)
  YKinPos=Len1*SIN(AKinPos)+Len2*SIN(AKinPos+BKinPos)
  CKinPos=AKinPos+BKinPos+WKinPos
  ZKinPos=Mtr4KinPos/Kv+Z0     ; For both tip and joint
  IF(BKinPos<0)                ; Left-armed case?
    SgnAcos=-1
  ELSE                         ; Right-armed case
    SgnAcos=1
  ENDIF
ELSE                           ; Not referenced; halt operation
  CS1RunTimeErr=1              ; Set run-time error bit
ENDIF
CLOSE
```

Inverse-Kinematic Algorithm to be Executed Repeatedly

```
&1 OPEN INVERSE                              ; Inverse kinematics for CS 1
CLEAR                                        ; Erase existing contents
IF(TipMode=1)                                ; Tip-mode programming?
  Temp1=XKinPos*XKinPos+YKinPos*YKinPos      ; X^2+Y^2
  Temp2=(Temp1-SumLenSqrd)/ProdOfLens        ; cos(E)
  IF (ABS(Temp2)<0.996)                      ; Valid soln w/ 5 deg margin?
    BKinPos=ACOS(Temp2)                      ; Elbow angle (deg)
    BKinPos=BKinPos*SgnAcos                  ; Select sign
    CosTerm=XKinPos                          ; X into cos argument for ATAN2
    SPlusQ=ATAN2(YKinPos)                    ; S+Q = ATAN2(Y,X)
    QPos=ACOS((Temp1+DifLenSqrd)/(2*Len1*SQRT(Temp1))) ; Q (deg)
    AKinPos=SPlusQ-QPos                      ; Shoulder angle (deg)
    WKinPos=CKinPos-AKinPos-BKinPos          ; Wrist angle (deg)
    Mtr1KinPos=AKinPos*Ks                    ; Shoulder position in counts
    Mtr2KinPos=BKinPos*Ke                    ; Elbow position in counts
    Mtr3KinPos=WKinPos*Kw                    ; Wrist position in counts
    Mtr4KinPos=(ZKinPos-Z0)*Kv               ; Vertical position in counts
  ELSE                                       ; Not valid, halt operation
    CS1RunTimeErr=1                          ; Set run-time error bit
  ENDIF
ELSE                                         ; Joint-mode programming
  Mtr1KinPos=AKinPos*Ks                      ; Shoulder position in counts
  Mtr2KinPos=BKinPos*Ke                      ; Elbow position in counts
  Mtr3KinPos=WKinPos*Kw                      ; Wrist position in counts
  Mtr4KinPos=(ZKinPos-Z0)*Kv                 ; Vertical position in counts
  ; Compute tip positions for future tip-mode commands
  XKinPos=Len1*COS(AKinPos)+Len2*COS(AKinPos+BKinPos)
  YKinPos=Len1*SIN(AKinPos)+Len2*SIN(AKinPos+BKinPos)
  CKinPos=AKinPos+BKinPos+WKinPos
ENDIF
CLOSE
```